Beeldverwerken Lab 2

Max Bos (10669027) & Philip Bouman (10668667)

April 16, 2017

This document contains answers to the theory questions and notes to the exercises belonging to Lab 2.

1 Convolution

Theory Answers

1.1 $f = \{1\ \underline{2}\ 1\}$
$g = \{0\ 0\ 0\ 0\ \underline{1}\ 1\ 1\ 1\ 1\}$
Flipping over g results in $g = \{1\ 1\ 1\ 1\ \underline{1}\ 0\ 0\ 0\ 0\}$
Handling the border cases by adding extra zeros to the locations where the convolution will have effect. Adding zeros to f, results in $f = \{1\ \underline{2}\ 1\ 0\ 0\ 0\ 0\}$
Applying convolution results in $f * g = \{1\ \underline{3}\ 4\ 4\ 4\ 3\ 1\ \}$

1.2 $f = \{0\ 0\ 0\ 0\ \underline{1}\ 1\ 1\ 1\ 1\}$
$g = \{1\ \underline{2}\ 1\}$
Flipping over g results in $g = \{1\ \underline{2}\ 1\}$
Handling the border cases by adding extra zeros to the locations where the convolution will have effect. Adding zeros to f, results in $f = \{0\ 0\ 0\ 0\ \underline{1}\ 1\ 1\ 1\ 1\ 0\}$
Applying convolution results in $f * g = \{1\ \underline{3}\ 4\ 4\ 4\ 3\ 1\ \}$

1.3 $f = \{0\ 0\ 0\ 0\ \underline{1}\ 1\ 1\ 1\ 1\}$
$g = \{\text{-}1\ \underline{1}\}$
Flipping over g results in $g = \{\underline{1}\ \text{-}1\}$
Handling the border cases by adding extra zeros to the locations where the convolution will have effect. Adding zeros to f, results in $f = \{0\ 0\ 0\ 0\ \underline{1}\ 1\ 1\ 1\ 1\ 0\}$
Applying convolution results in $f * g = \{\text{-}1\ \underline{0}\ 0\ 0\ 0\ 1\ \}$

1.4

$$(f(x) * g(x)) = \sum_{y \in E} f(y)g(x-y)$$
$$\text{Substitute }, u = x - y$$
$$x - y = u$$
$$(f * g)(x) = \sum_{y \in E} f(x-u)g(u)$$
$$(f * g)(x) = \sum_{y \in E} g(u)f(x-u)$$

Therefore it is commutative because we turned around x.

1.5

Convolution is associative when:

$$(f * (g * h))(x) = ((f * g) * h)(x)$$
$$= \sum_{y \in E} f(y)(g * h)(x-y)$$
$$= \sum_{y \in E} f(y) \sum_{z \in E} g(z)h((x-y)-z)$$
$$= \sum_{y \in E} \sum_{z \in E} f(y)g(z)h((x-y)-z)$$
$$= \sum_{y \in E} \sum_{z \in E} f(y)f((y+z)-y)h(x-(y+z))$$
$$\text{Subtitute } (y+z) \rightarrow u$$
$$= \sum_{u \in E} \sum_{y \in E} f(y)g(u-y)h(x-u)$$
$$= \sum_{u \in E} (f * g)(u)h(x-u)$$
$$= ((f * g) * h)(x)$$

1.1 Convolution Examples and Implementation

1.6 The convolution kernel for the identity operation on an image: $\left\{\underline{1}\right\}$

1.7 The convolution kernel for multiplying the image intensity by 3: $\left\{\underline{3}\right\}$

1.8 The kernel for translating the image over the vector $[-3, 1]^T$:
$\begin{Bmatrix} 0 & 0 & 0 & \underline{0} \\ 1 & 0 & 0 & 0 \end{Bmatrix}$

1.9 Impossible. Rotating an image over an arbitrary angle around the origin is not possible by convolution, as convolution performs an operation independant of the point it is performing it on.

1.10 The kernel for taking the average in a 3x3 neighbourhood:
$\frac{1}{9}\begin{Bmatrix} 1 & 1 & 1 \\ 1 & \underline{1} & 1 \\ 1 & 1 & 1 \end{Bmatrix}$

1.11 Impossible. Taking the median in a 3x3 is not possible with a convolution. It is not a translation invariant linear operation.

1.12 Impossible. Calculating the minimum in a 5x5 neighbourhood is not possible with a convolution. The values in the neighbourhood should be somehow compared to calculate the lowest one. But this is not a translation invariant linear operation. Therefore it can not be written as a convolution.

1.13 The kernel for performing motion blur, as if the camera moved horizontally to the right over 5 pixels during recording:
$\frac{1}{5}\begin{Bmatrix} 1 & 1 & 1 & 1 & \underline{1} \end{Bmatrix}$

1.14 The kernel for performing optical blur:
$\frac{20}{3}\begin{Bmatrix} 0.016 & 0.017 & 0.016 \\ 0.017 & \underline{0.018} & 0.017 \\ 0.016 & 0.017 & 0.016 \end{Bmatrix}$

1.15 The kernel for performing unsharp masking of an image:
$\begin{Bmatrix} -0.1067 & -0.1133 & -0.1067 \\ -0.1133 & 1.8800 & -0.1133 \\ -0.1067 & -0.1133 & -0.1067 \end{Bmatrix}$

1.16 The kernel for taking an approximation of the derivative in the x-direction:
$\frac{1}{2}\begin{Bmatrix} 1 & \underline{0} & -1 \end{Bmatrix}$

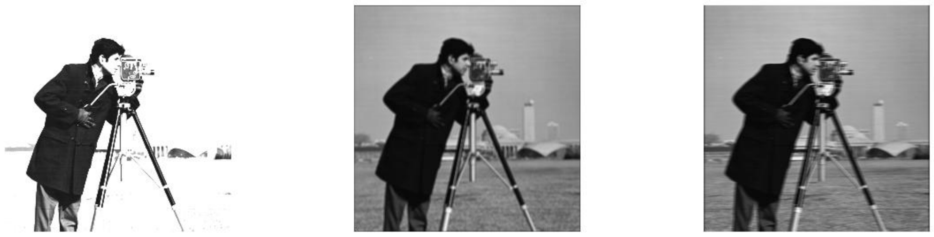
1.17 The kernel for taking the second derivative in the x-direction:
$\frac{1}{4}\begin{Bmatrix} 1 & 0 & \underline{-2} & 0 & 1 \end{Bmatrix}$

1.18 Impossible. Zooming in requires the addition of extra lines of zeros. Adding these is not a translation invariant linear operation and therefore can not be represented by a convolution.

1.19 Impossible. It is impossible to create a single kernel that can set pixel values to different constant values based on a condition.

Exercise Notes

The three figures above show three different filter operations on an image.



(a) Multiplying image intensity by 3 (b) The average in a 3x3 neighborhood (c) Motion blur, 5 pixels horizontal to the right

Figure 1: Three convolution operations on an image

1.2 Gaussian Convolution

1.20 The Gaussian has smooth derivatives of arbitrary order, because taking the derivative is a translation invariant linear operation.

1.21 'Taking a derivative' should be a convolution, as it enables the ability to combine this operation with other convolution operations. Taking the derivate is a translation invariant linear operation and therefore should be represented as a convolution.

1.22 The derivative of an image at scale σ can be computed by convolution with the derivative of a Gaussian kernel of that scale. Commutativity and associativity are involved in the following way:
$\partial(f * G\sigma) = D * (f * G\sigma) = f * (D * G\sigma) = f * (\partial G\sigma)$.

1.23

$$\begin{aligned}
G_\sigma(x) &= \frac{1}{(\sqrt{2\pi}\sigma)^2} e^{-\frac{x^2}{2\sigma^2}} \\
\frac{\partial(G\sigma)}{\partial x} &= \frac{1}{(\sqrt{2\pi}\sigma)^2} \frac{\partial}{\partial x} e^{-\frac{x^2}{2\sigma^2}} \\
&= \frac{1}{(\sqrt{2\pi}\sigma)^2} e^{-\frac{x^2}{2\sigma^2}} \frac{-x}{\sigma^2} \\
&= G_\sigma(x) \frac{-x}{\sigma^2} \\
&= -\frac{x}{\sigma^2} G_\sigma(x)
\end{aligned}$$

1.24 Using the product rule with $\frac{-x}{\sigma^2}$ and $e^{-\frac{|x|^2}{2\sigma^2}}$, results in:
$\frac{-1}{\sigma^2} \cdot e^{-\frac{|x|^2}{2\sigma^2}} + \frac{-x}{\sigma^2} \cdot \frac{-x}{\sigma^2}$
$\frac{\partial^2 G}{\partial x^2} = (\frac{x^2}{\sigma^4} - \frac{1}{\sigma^2} G_\sigma(x)$

1.25 $G_\sigma * G_\tau = G_{\sqrt{\sigma^2+\tau^2}}$

1.26

$$G_\sigma(x) = \frac{1}{\sqrt{2\pi}\sigma} e^{-\frac{x^2}{2\sigma^2}}$$
$$G_\sigma(y) = \frac{1}{\sqrt{2\pi}\sigma} e^{-\frac{y^2}{2\sigma^2}}$$
$$G_\sigma(x)G_\sigma(y) = \frac{1}{\sqrt{2\pi}\sigma} e^{-\frac{x^2}{2\sigma^2}} \frac{1}{\sqrt{2\pi}\sigma} e^{-\frac{y^2}{2\sigma^2}}$$
$$= \frac{1}{(2\pi\sigma)^2} e^{-\frac{x^2+y^2}{2\sigma^2}}$$
$$= G_\sigma$$

The scales of those smoothings are σ

1.27 $\frac{\delta(\frac{\delta G_\sigma(x)}{\delta x})}{\delta y}(y) = \frac{\delta G_\sigma(x,y)}{\delta x \delta y}$

2 Implementation of Gaussian derivatives

2.1 Exercise Notes

2.1 We believe $ceil(3 * sigma) = a$ is an appropriate radius for the kernel. Which results in a grid size of (a*2+1)x(a*2+1). This is based on the three-sigma rule. This rule states that 99.7 of the data are within 3 standard deviations of the mean.

2.2 It is expected that the result of the overall sum of the Gaussian kernel is 1. Since, the Gaussian distribution is a probability distribution, which is normalized, so the sum over all values of the kernel gives a probability of 1.

2.3

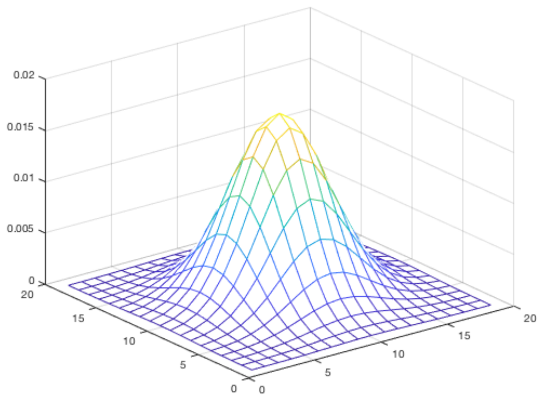


Figure 2: Plot of the Gaussian kernel

2.4 The physical unit of the scale parameter sigma is pixel unit.

2.6 The order of the computational complexity seems to be quadratic.

2.9 The order of computational complexity seems to be quadratic using the separability property.

2.10 The code for gD function can be seen below

```
function result = gD(image, sigma, xorder, yorder)
    radius = round(3*sigma);
    x = -radius:radius;
    g = Gauss1(sigma);
    dx = -x .* g ./ sigma^2;
    dxx = ((x .^2 ./ sigma^4) - (1/sigma^2)) .* g;
    if xorder == 0
        image = imfilter(image, g, 'conv', 'replicate');
    end
    if xorder == 1
        image = imfilter(image, dx, 'conv', 'replicate');
    end
    if xorder == 2
        image = imfilter(image, dxx, 'conv', 'replicate');
    end
    if yorder == 0
        image = imfilter(image, g', 'conv', 'replicate');
    end
    if yorder == 1
        image = imfilter(image, dx', 'conv', 'replicate');
    end
    if yorder == 2
        image = imfilter(image, dxx', 'conv', 'replicate');
    end
    result = image;
end
```

2.11

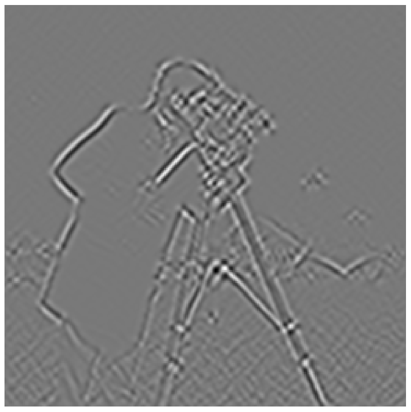


Figure 3: 2-jet of cameraman.tif

3 The Canny Edge Detector

3.1 Exercise Notes

3.1

$$f(x,y) = Asin(Vx) + Bcos(Wy)$$

Finding the derivatives of above function involves using implicit differentiation and the derivatives of regular trigonometric functions.

$$f_x = AVcos(Vx)$$
$$f_y = -BWsin(Wy)$$
$$f_{xx} = -AV^2sin(Vx)$$
$$f_{yy} = -BW^2cos(Wy)$$
$$f_{xy} = 0$$

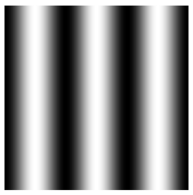


Figure 4: Partial derivative of $f(x,y)$ with respect to x

3.2

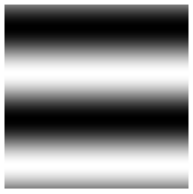


Figure 5: Partial derivative of $f(x,y)$ with respect to y

3.3

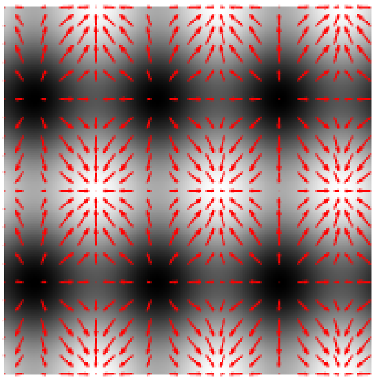


Figure 6: Image overlayed with vectors

3.4

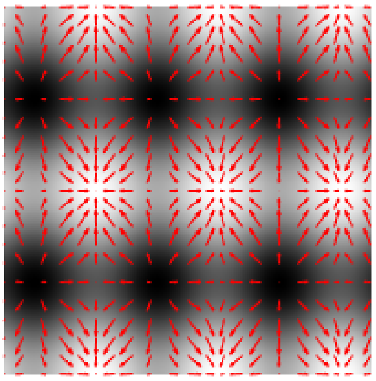


Figure 7: Image overlayed with vectors using gD function

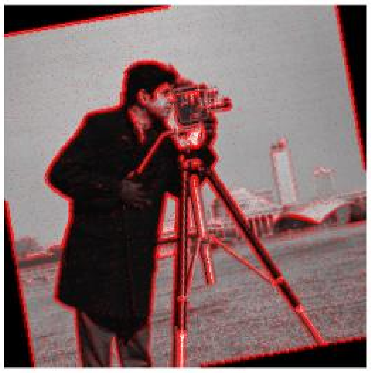


Figure 8: Plot of the rotated image with gradient vectors

3.5

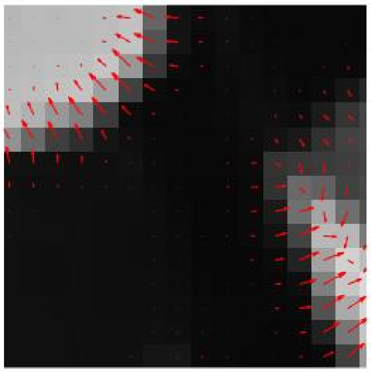


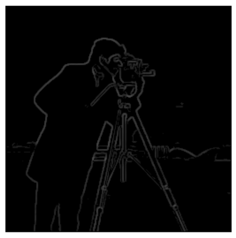
Figure 9: Zoomed in on plot of the rotated image with gradient vectors

3.6

$$f_x = AVcos(Vx)$$
$$f_y = -BWsin(Wy)$$
$$f_{xx} = -AV^2sin(Vx)$$
$$f_{yy} = -BW^2cos(Wy)$$
$$f_{xy} = 0$$
$$f_w = \sqrt{A^2v^2cos^2(vx) + B^2W^2sin^2(Wy)}$$
$$f_{ww} = \frac{-A^3V^4sin(Vx)cos^2(Vx) + -B^3W^4sin^2(Wy)cos(Wy))}{A^2v^2cos^2(vx) + B^2W^2sin^2(Wy)}$$

3.7 See the code in matlab.

3.8 See Fig. 10 for the result of the test on the implementation of the Canny Edge operation on cameraman.tif.


(a) Result image of Canny Edge operation

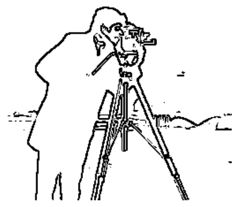

(b) Inverted result image of Canny Edge operation

Figure 10: Canny Edge operation on cameraman.tif

3.9 A chessboard is shown, and a result that of that chessboard using our corner detector.

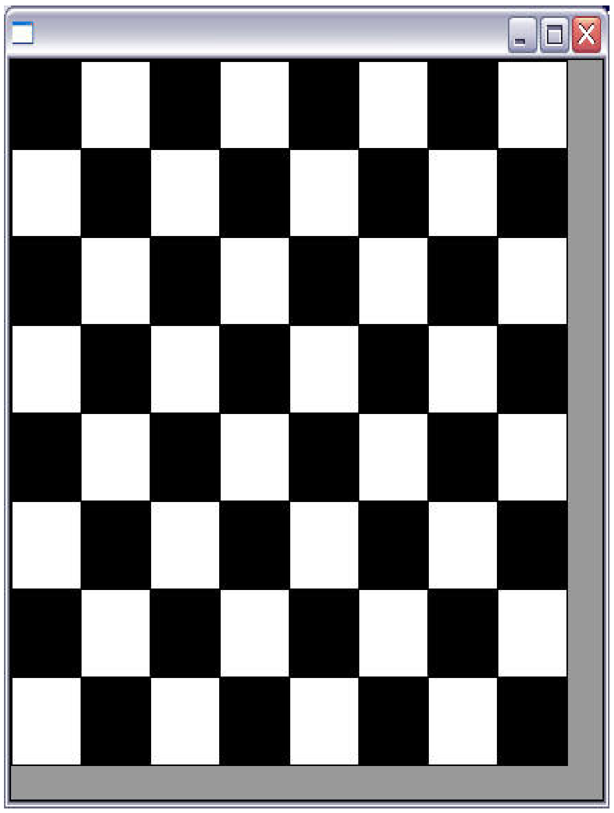


Figure 11: Picture of a chessboard

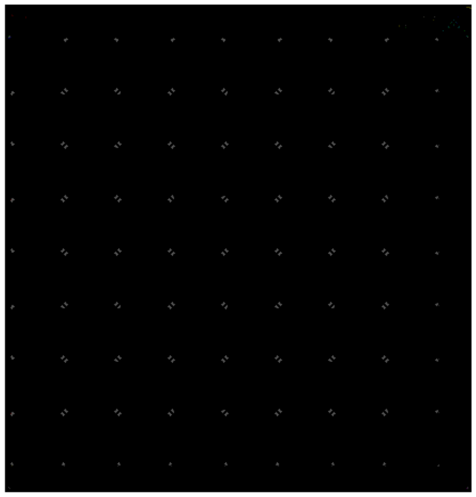


Figure 12: The result of the corner detector